

Transformation von XML-Dokumenten

Heutige Vorlesung

- Warum XML-Dokumente transformieren?
- Transformation von XML-Dokumenten mit XSLT
- XPath (wird von XSLT benutzt)
- Vor- und Nachteile von XSLT

Warum XML-Dokumente transformieren?

Trennung Inhalt und Präsentation

- XML trennt Inhalt von Präsentation (Layout).
- sollen XML-Inhalte präsentiert werden, müssen diese transformiert werden:

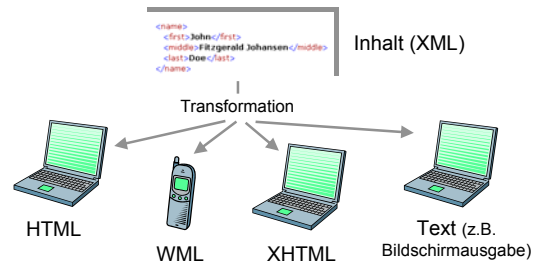
XML-Inhalt → Layout

Inhaltliche Transformationen

- interne Geschäftsdaten mit XML repräsentiert
- unterschiedliche Sichten (views) auf XML-Inhalte erfordern Transformationen:

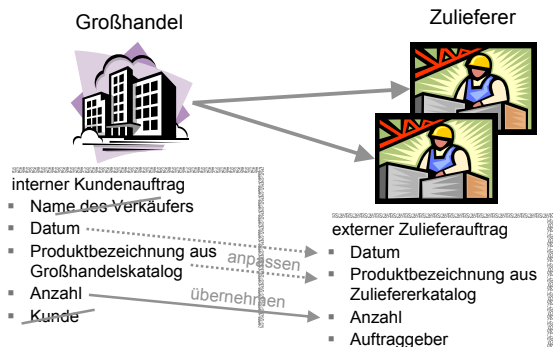
XML-Inhalt → XML-Inhalt

XML-Inhalt → Layout



- **Multi-Delivery:** ein Inhalt, verschiedene Layouts
- **Beachte:** XHTML, WML ⊂ XML

XML-Inhalt → XML-Inhalt



XML-Inhalt → XML-Inhalt

Kundenauftrag

```
<?xml version="1.0"?>
<order>
  <salesperson>John Doe</salesperson>
  <item>Production-Class Widget</item>
  <quantity>16</quantity>
  <date>
    <month>1</month>
    <day>13</day>
    <year>2000</year>
  </date>
  <customer>Sally Finkelstein</customer>
</order>
```

Zulieferauftrag

andere Sicht (view) auf XML-Inhalt

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <date>2000/1/13</date>
  <customer>Company A</customer>
  <item>
    <part-number>E16-25A</part-number>
    <description>Production-Class Widget</description>
    <quantity>16</quantity>
  </item>
</order>
```

XSLT

Was ist XSLT?

- XSLT = Extensible Stylesheet Language Transformations
- W3C-Standard von 1999
- spezielle Programmiersprache zur Transformation von XML-Dokumenten
- erlaubt XML-Dokumente in beliebige Textformate zu Transformieren:
XML → XML / HTML / XHTML / WML / Text etc.
- XSLT-Programme (*stylesheets*) haben XML-Syntax
→ plattformunabhängig

Programmierparadigma

XSLT-Programm (*stylesheet*)

= Menge von Transformationsregeln

Transformationsregel (*template*)

- Erzeuge aus Unterstruktur X im Ursprungsdokument Y im Ergebnisdokument!

Beispiel:

```
<xsl:template match="order/item">
  <p>Item encountered</p>
</xsl:template>
```

```
<order>...
  <item>...</item>...
</order>
↓
<p>Item encountered</p>
```

- XSLT benutzt zur Identifizierung von Unterstrukturen W3C-Standard XPath.

Beispiel

```
<xsl:template match="order/item">
  <p>Item encountered</p>
</xsl:template>

<?xml version="1.0"?>
<order>
  <salesperson>John Doe</salesperson>
  <item>Production-Class Widget</item>
  <quantity>16</quantity>
  <date>...</date>
  <customer>Sally Finkelstein</customer>
</order>

↓

<p>Item encountered</p>
```

Template

Ursprungsdokument
(*source tree*)

Ergebnisdokument

XSLT-Stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="order/item">
    ...
  </xsl:template>
</xsl:stylesheet>
```

- XML-Dokument
- Dokument-Wurzel: entweder stylesheet oder transform aus entsprechendem W3C-Namensraum.
- stylesheet und transform gleichbedeutend

XPath

XPath



- W3C-Standard von 1999
- Standard zum Zugreifen beliebiger Teile eines XML-Dokumentes
- wird von XSLT benutzt
- Adressierungspfad eines Dateisystems ähnlich:
z.B. /order/item

Zugrundelegendes Dokumentenmodell



- XML-Dokument wird als Baum mit Elementen, Attributen und Text-Inhalten als Knoten interpretiert (wie in DOM).
- virtuelle Dokument-Wurzel wird angenommen (ebenfalls wie in DOM):
durch "/" repräsentiert (links von "/" steht nichts)
- Wurzel-Element daher immer Kind von "/":
z.B. /order

Zugriff auf Elemente und Attribute



- Elemente werden einfach mit ihrem Namen identifiziert:
z.B. **order** oder **order/item**
- Attribute werden mit "@name" identifiziert:
z.B. **@id** oder **order/@id**

Absolute und relative Pfade



absolute Pfade

- beginnen mit "/"
- z.B. /order/item
→ lesen: „Folge dem Pfad von der Dokument-Wurzel zu Kind-Elementen order und von dort aus zu Kind-Elementen item!“

relative Pfade

- beginnen mit einem Element oder Attribut
- z.B. order/item
← lesen: „item-Elemente, die Kind eines Elementes order sind“
- Element order an beliebiger Stelle des XML-Dokumentes

Pfad-Ausdrücke



- aktueller Knoten
- .. Eltern-Knoten des aktuellen Knotens
- * beliebiges Kind-Element des aktuellen Knotens
- @* beliebiges Attribut des aktuellen Knotens
- // überspringt ≥ 0 Hierarchie-Ebenen vom aktuellen Knoten ausgehend
- | Auswahl (Vereinigung)
- **Beispiel:** *|@*
„Kind-Elemente und Attribute des aktuellen Knotens“

Kontext-Knoten



- Pfade werden immer bzgl. eines bestimmten Kontextes ausgewertet:
Element-, Attribut- oder Text-Knoten
- **Beispiel:**

```
<xsl:template match="p">
  <DIV>
    <xsl:value-of select="." />
  </DIV>
</xsl:template>
```
- Kontext-Knoten = Knoten, auf den das Template angewandt wird (hier: ein p-Element)

Aktueller Knoten "."

- zunächst der Kontext-Knoten
- entlang des Pfades verändert sich der aktuelle Knoten entsprechend, z.B.:
 - order/..
↑ "." = Kontext-Knoten K
 - order/..
↑ "." = Kind-Element von K mit Namen order

Filter: Randbedingungen für Pfade

- order/item[@item-id = 'E16-25A']
„item-Elemente, die Kind von order sind und Attribut item-id mit Wert 'E16-25A' haben“
- können an beliebiger Stelle in einem Pfad vorkommen:
order[@order-id = '4711']/item
- können Vielzahl von Funktionen, wie z.B. text() benutzen:
order/item[text(.) = 'E16-25A']
- text(.) liefert Text-Inhalt des aktuellen Knotens
- können Boolesche Ausdrücke enthalten:
order/item[text(.) = 'E16-25A' or @item-id = 'E16-25A']

XSLT-Templates

Inhalte erzeugen

- Grundprinzip: „Suche im Ursprungsdokument bestimmte Unterstruktur X und **erzeuge hieraus im Ergebnisdokument Y!**“
- zwei verschiedene Möglichkeiten, Y zu erzeugen:
 1. Neue Inhalte erzeugen, also Text, Elemente oder Attribute.
 2. Inhalte von X nach Y übertragen.
- beide Möglichkeiten beliebig miteinander kombinierbar

Neue Inhalte erzeugen

- Templates können alle drei Kategorien von XML-Inhalten erzeugen: Text, Elemente und Attribute.
- Jeweils einfach normale XML-Syntax verwenden:

```
<xsl:template match="order/item">  
  <neu attr="value">neuer Text</neu>  
</xsl:template>
```

- **Beachte:** Stylesheets müssen wohlgeformte XML-Syntax haben, daher z.B. nicht erlaubt:

```
<xsl:template match="order/item">  
  <br>neuer Text  
</xsl:template>
```

Beispiel

Template

```
<xsl:template match="order/item">  
  <neu attr="value">neuer Text</neu>  
</xsl:template>
```

Ursprungsdokument

```
<?xml version="1.0"?>  
<order>  
  <salesperson>John Doe</salesperson>  
  <item>Production-Class Widget</item>  
  <quantity>16</quantity>  
  <date>...</date>  
  <customer>Sally Finkelstein</customer>  
</order>
```

Ergebnisdokument

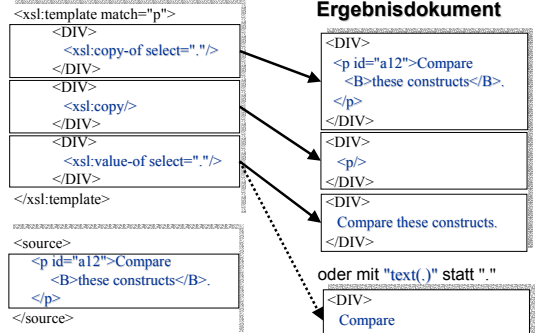
```
<neu attr="value">  
  neuer Text  
</neu>
```

Inhalte übertragen

drei Möglichkeiten, Inhalte zu übertragen:

- `<xsl:copy-of select="."/>`: Kopiert aktuellen Teilbaum.
- **aktueller Teilbaum**: Baum, der vom aktuellen Knoten aufgespannt wird, einschließlich aller Text-Inhalte und Attribute
- `<xsl:copy/>`: Kopiert aktuellen Knoten *ohne* Attribute, Kind-Elemente und Text-Inhalt.
Kopiert also nur Wurzel des aktuellen Teilbaums.
- `<xsl:value-of select="."/>`: Extrahiert alle Text-Inhalte aus aktuellem Teilbaum.

Beispiel

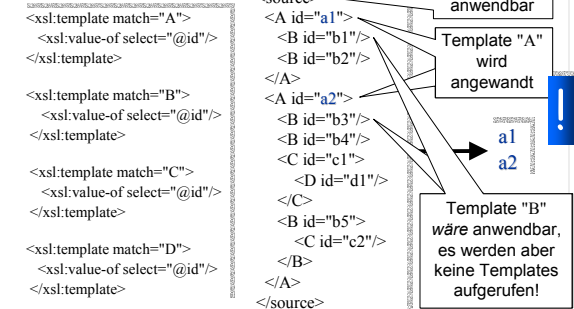


XSLT-Prozessor

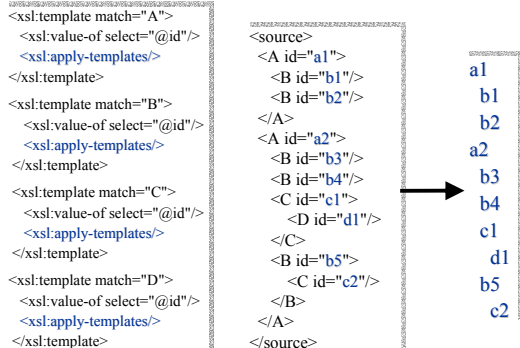
- ausgehend von der Dokument-Wurzel wird versucht, auf einen Knoten K im Ursprungsdokument ein Template anzuwenden:
- Zuerst werden alle Templates identifiziert, die auf K anwendbar sind.
- Ist *genau ein* Template anwendbar, dann wird dieses angewandt.
- Sind *mehre* Templates anwendbar, dann wird das *speziellste* angewandt: z.B. ist `"/order` spezieller als `"/*`.
- Ist *kein* Template anwendbar, so wird versucht, auf *jedes* Kind-Element von K ein Template anzuwenden.

Beispiel

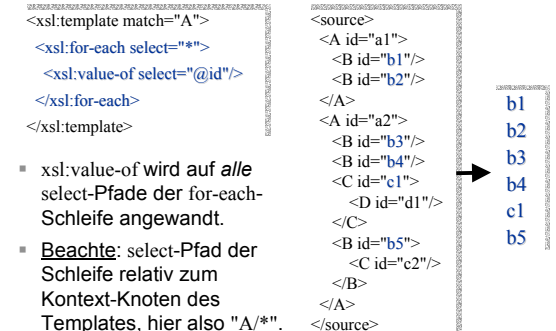
Stylesheet



Templates mit Rekursion



Iteration statt Rekursion



Vordefinierte Templates



zwei vordefinierte Templates:

- Template 1: realisiert rekursiven Aufruf des Prozessors, wenn kein Template auf aktuellen Knoten anwendbar ist.
 - Template 2: kopiert Text-Inhalte und Attribut-Werte in das Ergebnisdokument.
- Leeres Stylesheet traversiert gesamtes Ursprungsdocument und extrahiert alle Text-Inhalte und Attribut-Werte.

Beide vordefinierten Templates können durch speziellere Templates überschrieben werden.

Erstes vordefinierte Template



```
<xsl:template match="*/" />
<xsl:apply-templates/>
</xsl:template>
```

1. wird zuerst auf Dokument-Wurzel "/" angewandt
2. ruft alle Templates (einschl. sich selbst) auf
3. wird auf alle Kind-Elemente "*" angewandt

versucht, alle Templates rekursiv auf Kind-Elemente anzuwenden, falls kein spezielleres Template wie `<xsl:template match="order">` anwendbar ist

Zweites vordefinierte Template



```
<xsl:template match="text()|@"*>
  <xsl:value-of select="."/>
</xsl:template>
```

- wird auf jeden Text-Knoten und jedes Attribut angewandt
- überträgt einfach Text-Inhalt bzw. Attribut-Wert in das Ergebnisdokument

Leeres Stylesheet



- Bei Stylesheet *ohne* Templates sind nur die beiden vordefinierten Templates aktiv:

```
<xsl:template match="*/" />
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="text()|@"*>
  <xsl:value-of select="."/>
</xsl:template>
```

- Gesamtes Ursprungsdocument wird traversiert, dabei werden Text-Inhalte und Attribut-Werte extrahiert:

```
<name>
  <first>John</first>
  <middle>Fitzgerald Johansen</middle>
  <last>Doe</last>
</name>
```

→ John Fitzgerald Johansen Doe

Identitäts-Stylesheet



- Stylesheet mit lediglich einem Template:

```
<xsl:template match="*/" />
  <xsl:copy>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

- wird auf jedes Element angewandt
- Vordefiniertes Template `<xsl:template match="*/" />` ist allgemeiner und wird deshalb überschrieben.
- Zusammen mit dem zweiten vordefinierten Template `<xsl:template match="text()|@"*>` wird gesamtes Ursprungsdocument kopiert.

Benannte Templates



- Templates können auch einen Namen haben:

```
<xsl:template match="/order/item" name="order-template">
  <p>Item encountered</p>
</xsl:template>
```

- Benannte Templates können gezielt mit

```
<xsl:call-template name="order-template"/>
```

aufgerufen werden.

Kontrollfluss: if

Beispiel:

```
<xsl:if test="number(employee/@FullSecurity)">
  <p><xsl:value-of select="employee/salary"/></p>
</xsl:if>
```

- überprüft, ob Attribut FullSecurity des Elementes employee einen Wert > 0 hat.

Kontrollfluss: choose

Beispiel:

```
<xsl:template match="item">
  <part-number>
    <xsl:choose>
      <xsl:when test="text() = 'Production-Class Widget'">
        E16-25A
      </xsl:when>
      <xsl:when test="text() = 'Economy-Class Widget'">
        E16-25B
      </xsl:when>
      <xsl:otherwise>00</xsl:otherwise>
    </xsl:choose>
  </part-number>
</xsl:template>
```

- Switch-Anweisung in Java ähnlich
- Abarbeitung von oben nach unten

Zugriff auf Datenbanken

```
<xsl:template match="item">
  <part-number>
    <xsl:choose>
      <xsl:when test="text() = 'Production-Class Widget'">
        E16-25A
      </xsl:when>
      <xsl:when test="text() = 'Economy-Class Widget'">
        E16-25B
      </xsl:when>
      <xsl:otherwise>00</xsl:otherwise>
    </xsl:choose>
  </part-number>
</xsl:template>
```

- Zugriff auf Datenbank wäre besser.
- In XSLT ist dies jedoch *nicht* möglich.

- xsl:import erlaubt jedoch importieren externer Templates.
- mögliche Lösung: Datenbank → Templates → xsl:import

Variablen

Beispiel:

```
<xsl:variable name="X">
  <xsl:value-of select="/name/first">
</xsl:variable>
```

- Deklariert Variable X und weist ihr den Text-Inhalt des Knotens /name/first zu.
- Beachte:** Initiale Zuweisung kann *nicht* überschrieben werden!
- Variablen können beliebige XML-Strukturen enthalten und als Parameter von Templates übergeben werden.

Mächtigkeit von XSLT

- Variablen machen Stylesheets zu einem mächtigen Termersetzungssystem mit unbeschränkten Registern.
- www.unidex.com/turing definiert universelle Turingmaschine in XSLT.
- Damit wird der IE zum vollwertigen Computer!
- Stylesheets tatsächlich berechnungsvollständig und damit eine vollwertige Programmiersprache (Kepser 2002).

→ Terminierung von Stylesheets kann nicht garantiert werden.

Principle of Least Power

When I designed HTML for the Web, I chose to avoid giving it more power than it absolutely needed - a "principle of least power," which I have stuck ever since. I could have used a language like Donald Knuth's "TeX," which though it looks like a markup language is in fact a programming language. It would allow you to express absolutely anything on the page, but would also have allowed Web pages that could crash, or loop forever (Tim Berner-Lees, 1999).

Verletzt XSLT dieses grundlegende Prinzip?

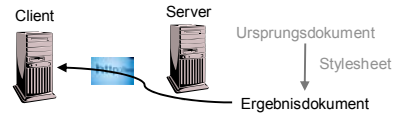
Verarbeitung von Stylesheets



Stylesheets können auf zwei Arten verarbeitet werden:

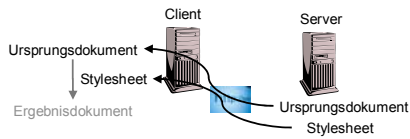
- auf dem Server
 - im Client
- Worin besteht der Unterschied?
jeweiligen Vor- und Nachteile

Verarbeitung auf dem Server



- Server wendet passendes Stylesheet auf Ursprungsdokument an.
- MSXML: `msxsl source stylesheet.xml -o output`
- Client bekommt nur Ergebnisdokument

Verarbeitung im Client



- Client bekommt Ursprungsdokument und passendes Stylesheet.
- Im Ursprungsdokument:
`<?xml-stylesheet type="text/xsl" href="stylesheet.xml"?>`
- Web-Browser wendet Stylesheet automatisch an und stellt Ergebnisdokument dar.

Vor- und Nachteile



Verarbeitung im Client

- + Transformationen auf Clients verteilt: spart Server-Ressourcen
- Ursprungsdokument sichtbar

XSLT: stellt sicher, dass Transformation im Web-Client ausgeführt werden kann.

Verarbeitung auf dem Server

- + Ursprungsdokument verdeckt
- alle Transformationen auf zentralen Server

XSLT: *nicht* unbedingt nötig, da Transformation auf eigenem Server durchgeführt wird.

Vorteile von XSLT



- + plattformunabhängig
- + relativ weit verbreitet
- + Verarbeitung in Web-Browsern möglich
- + Standard-Transformationen (wie XML → HTML) einfach zu realisieren.
- + Nicht nur HTML, sondern beliebige andere Sprachen können erzeugt werden.
- + extrem mächtig

Nachteile von XSLT



- Entwickler müssen *speziell* für die Transformation von XML-Dokumenten neue Programmiersprache lernen.
- Anbindung von Datenbanken umständlich
- komplexe Transformationen, wie Join von zwei Relationen nur umständlich zu realisieren.
- Terminierung kann nicht garantiert werden.

Fazit: XSLT nur für Standard-Transformationen verwenden!

Wie geht es weiter?



- ☑ Warum XML-Dokumente transformieren?
- ☑ XSLT und XPath

heute 16:15 Tutorium

- Beispiel XSLT-Stylesheet
- Übungsblatt 3

nächste Woche (2.6.)

- XML & Datenbanken